Building Linguistically Motivated Speech Recognisers with Regulus

Manny Rayner
Beth Ann Hockey
Pierrette Bouillon

Regulus community

Johan Bos	David Carter
Nikos Chatzichrisafis	John Dowding
Dominique Estival	Genevieve Gorrell
Kyoko Kanzaki	Maria Kaplanidou
Ian Lewin	Yukie Nakao
Marianne Santaholma	Marianne Starlander

Outline

- Overview
- Compiling unification grammars into speech recognizers
- Comparison of Regulus and other methods
- Using Regulus

Recognizer = Acoustic model + Grammar

- (Declarative information in recognizer…)
- Acoustic model describes the structure of the sounds
  - Beyond the scope of this talk…
- We are interested in grammars

The role of grammars in a spoken dialogue system

- Grammars provide both
  - Filter on recognition: defines what system can hear, reduces search
  - Defines what semantic structures are associated with which pieces of language

Standard methods for building speech recognition grammars

- Statistical language models
  - Need a lot of training data
  - Don’t produce any data structures
- Hand coded context-free grammars (CFGs)
  - Labor intensive
  - Hard to maintain

Unification grammars (UG)

- "Parameterised CFGs"
- Example:
  - CFG representation
    - NP_SG → D_SG, N_SG
    - NP_PL → D_PL, N_PL
  - UG representation
    - NP:[num=X] → D:[num=X], N:[num=X]

Previous work: compilers

- Gemini (Moore, Gawron, Dowding)
  – CommandTalk, Personal Satellite Assistant, WITAS…
- EPFL compiler (Chapellier, Rajman et al)
  – EPFL directory inquiry system
- HPSG2CFG (Kiefer and Krieger)
  – Not used in implemented speech system (?)
- Regulus 1 (Rayner, Hockey, Dowding)
  – On/Off House, MedSLT 1, Franco
- Uniance (Bos)
  – IBL

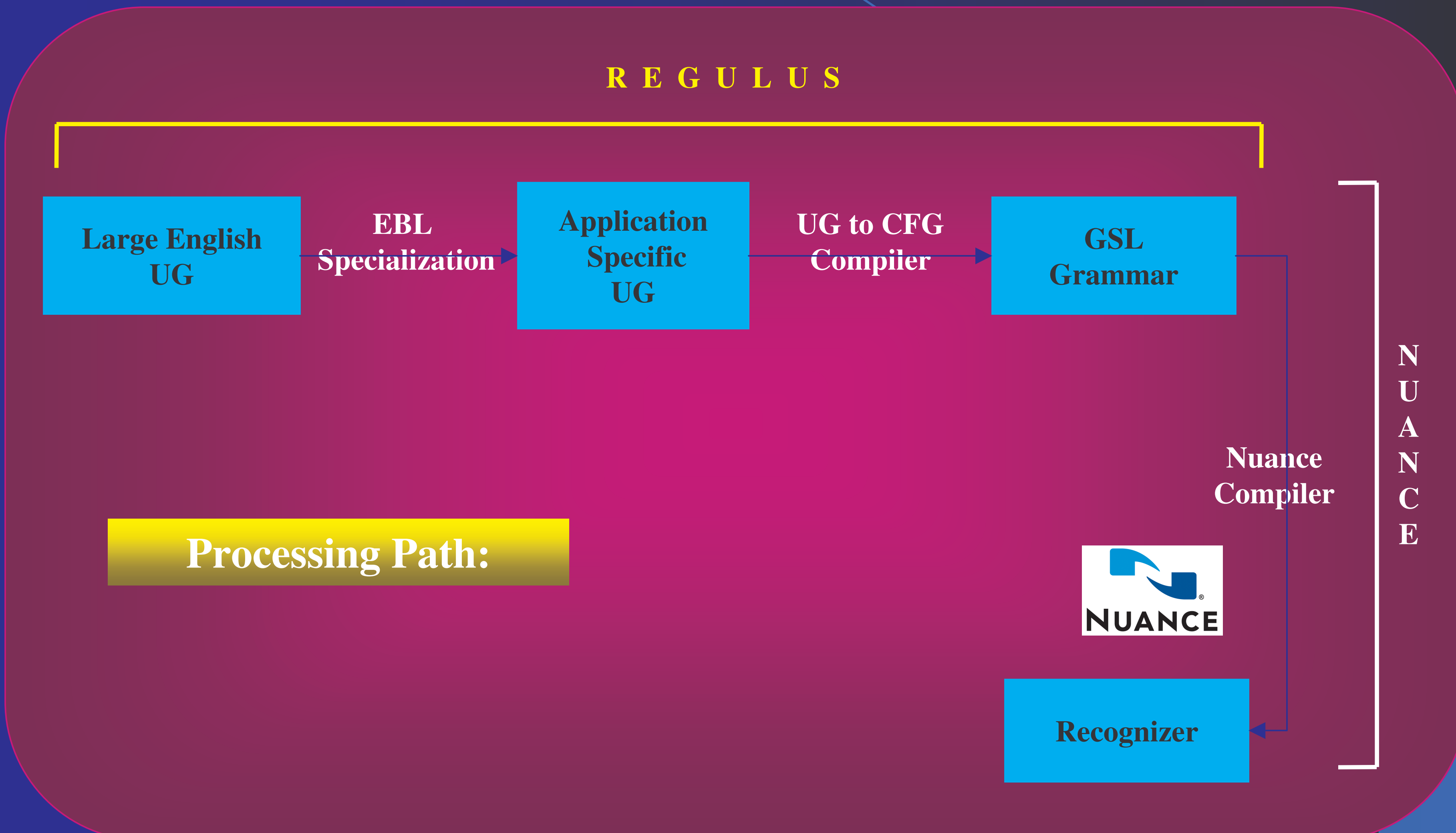
Limitations of previous work

- Domain-specific unification grammars
  - (All systems except Kiefer & Krieger)
  - New grammar needed for each domain
  - Not easy to port grammars
- (Kiefer and Krieger)
  - Can compile general unification grammars
  - Unclear whether resulting CFG grammar can be used in recognizer

The Regulus program

- Write a single *application-independent* unification grammar
- Derive application-specific unification grammars using example-based methods and small corpora
- Compile specialized unification grammars into CFG language models
- Good performance of recognizers on real tasks

The Regulus picture



The Regulus system

- Open Source platform compatible with Nuance recognizer
- Integrated development environment
- Has been used to build several non-trivial apps
  - Clarissa astronaut assistant, MedSLT translator

Key questions about Regulus

- How does it work?
- How does it scale up?
- How does it compare to alternatives?
- How do you use it?

Outline

- Overview
- Compiling unification grammars into speech recognizers
  - Unification grammar → CFG
  - Approximation using grammar specialization
  - Scalability
- Comparison of Regulus and other methods
- Using Regulus

Unification grammar → CFG

- Basic idea
  – Exhaustively expand rules
  – Filter results to remove useless rules
- Refinements
  – Efficient filtering
  – Interleaving of expansion and filtering
  – Pre-processing of grammar
  – Grammar compaction
  – Semantics

Exhaustive expansion

- Each feature in unification grammar has defined finite range of values
- Instantiate each feature to each of its possible values
- Problem: combinatoric explosion

Example unification grammar

Range of values for num: {sg, pl}

SIGMA:[] → NP:[num=X]
NP:[num=X] → D:[num=X], N:[num=X]
D:[num=sg] → this
D:[num=pl] → these
N:[num=sg] → cat
N:[num=pl] → cats

Expanded grammar

SIGMA:[] → NP:[num=sg]
SIGMA:[] → NP:[num=pl]
NP:[num=sg] → D:[num=sg], N:[num=sg]
NP:[num=pl] → D:[num=pl], N:[num=pl]
D:[num=sg] → this
D:[num=pl] → these
N:[num=sg] → cat
N:[num=pl] → cats

(Normal CFG notation…)

SIGMA → NP_SG
SIGMA → NP_PL
NP_SG → D_SG, N_SG
NP_PL → D_PL, N_PL
D_SG → this
D_PL → these
N_SG → cat
N_PL → cats

Filtering

- Some expanded rules may be irrelevant
- Top down filtering
  – Rules irrelevant because they don’t connect to the top-level rule
- Bottom up filtering
  – Rules irrelevant because they don’t connect to the lexicon

Example of top-down filtering

Range of values for num: {sg, pl}

SIGMA:[] → NP:[num=sg]
NP:[num=X] → D:[num=X], N:[num=X]
D:[num=sg] → this
D:[num=pl] → these
N:[num=sg] → cat
N:[num=pl] → cats

Expanded grammar

SIGMA:[] → NP:[num=sg]
NP:[num=sg] → D:[num=sg], N:[num=sg]
NP:[num=pl] → D:[num=pl], N:[num=pl]
D:[num=sg] → this
D:[num=pl] → these
N:[num=sg] → cat
N:[num=pl] → cats

Filtered grammar

SIGMA:[] → NP:[num=sg]
NP:[num=sg] → D:[num=sg], N:[num=sg]
D:[num=sg] → this
N:[num=sg] → cat

Example of bottom-up filtering

Range of values for num: {sg, pl}

SIGMA:[] → NP:[num=X]
NP:[num=X] → D:[num=X], N:[num=X]
D:[num=sg] → this
N:[num=sg] → cat

Expanded grammar

SIGMA:[] → NP:[num=sg]
SIGMA:[] → NP:[num=pl]
NP:[num=sg] → D:[num=sg], N:[num=sg]
NP:[num=pl] → D:[num=pl], N:[num=pl]
D:[num=sg] → this
N:[num=sg] → cat

Filtered grammar

SIGMA:[] → NP:[num=sg]
NP:[num=sg] → D:[num=sg], N:[num=sg]
D:[num=sg] → this
N:[num=sg] → cat

Efficient filtering

- Want filtering time to be linear in #rules
- Top-down filtering is easy
  - Just propagate down from the root category
- Bottom-up is less trivial
  - Obvious algorithm is quadratic time

Linear-time bottom-up filtering

- Linear-time bottom-up filtering is possible
- Corollary of result by Dowling & Galliers
  – Good concise explanation in Russell & Norvig
- Key idea: make bottom-up filtering into a marker-passing process
- Actually not quite linear in our implementation … O(n log(n))

Bottom-up filtering method

- "Supported non-terminal N"
  - Def: can generate at least one string from N
  - Base case: there is a lexical entry for N
- "Missing support for rule R"
  - Def: # unsupported non-terminals in RHS of R
  - Decrement missing support if non-terminal becomes supported
  - Rule is supported if missing support = 0
    - Non-terminal on LHS becomes supported
- Algorithm
  - Percolate supported non-terminals upwards

Example

1 SIGMA:[] → NP:[num=sg]
1 SIGMA:[] → NP:[num=pl]
2 NP:[num=sg] → D:[num=sg], N:[num=sg]
2 NP:[num=pl] → D:[num=pl], N:[num=pl]
0 D:[num=sg] → this
0 N:[num=sg] → cat

Black figures show missing support

Example

1 SIGMA:[] → NP:[num=sg]
1 SIGMA:[] → NP:[num=pl]
0 NP:[num=sg] → D:[num=sg], N:[num=sg]
2 NP:[num=pl] → D:[num=pl], N:[num=pl]
0 D:[num=sg] → this
0 N:[num=sg] → cat

Black figures show missing support

Example

0 SIGMA:[] → NP:[num=sg]
1 SIGMA:[] → NP:[num=pl]
0 NP:[num=sg] → D:[num=sg], N:[num=sg]
2 NP:[num=pl] → D:[num=pl], N:[num=pl]
0 D:[num=sg] → this
0 N:[num=sg] → cat

Black figures show missing support

Timings for bottom-up filtering

# Rules	Time/Rule (msecs)
1132	0.06 msecs/rule
2966	0.05 msecs/rule
4037	0.06 msecs/rule
18880	0.06 msecs/rule
117811	0.07 msecs/rule

Interleaving of expansion and filtering

- #Expanded rules exponential in #features
- May run out of space before we can filter
- Solution: interleave expansion and filtering
  – Expand using subset of features
  – Filter
  – Iterate until all features have been expanded

Importance of interleaved expansion and filtering

- Try compiling without interleaving
- Increase number of features in grammar

#Features	#Rules before filtering	#Rules after filtering	Time (secs) No interleaving
10	412	364	0.1
20	771	388	0.2
30	2027	468	0.7
36	56849	1082	5.3
38	210933	1086	99.9
40	(exceeded resource limits)		

Pre-processing of grammars

- Can reduce size of expanded CFG grammar by pre-processing unification grammar
- Two transforms currently used
  - Singleton variable elimination
  - Binarization

Singleton variable elimination

- "Singleton variables" can be optimized
- Example: transitive VP rule

VP:[num=SubjNum] →
    V:[num=SubjNum, subcat=trans],
    NP:[num=ObjNum, gender=Gen]

Expands to 2 x 2 x 2 = 8 CFG rules

ObjNum and Gen are singleton variables

Singleton variable elimination

Transformed version:

VP:[num=SubjNum] →
V:[num=SubjNum, subcat=trans],
NP:[num=any, gender=any]

NP:[num=any, gender=any] →
NP:[num=Num, gender=Gen]

Expands to 2 + 2 x 2 = 6 CFG rules

Binarization

- Rules with many daughters cause problems
  - Number of generated CFG rules is exponential in number of daughters
- Solution: apply a binarization transform
  - In binarized grammar, rules have ≤ 2 daughters

Binarization

VP:[num=SubjNum] $\rightarrow$
V:[num=SubjNum, subcat=ditrans],
NP:[num=IndObjNum],
NP:[num=ObjNum]

$\Rightarrow$

VP:[num=SubjNum] $\rightarrow$
V:[num=SubjNum, subcat=ditrans],
TMP1:[num1=IndObjNum, num2=ObjNum]

TMP1:[num1=IndObjNum, num2=ObjNum] $\rightarrow$
NP:[num=IndObjNum],
NP:[num=ObjNum]

Grammar compaction

- Can also apply CFG → CFG transforms to simplify resulting grammar
- Probabilistic training of CFG grammar works better on smaller grammar
  - Fewer rules means fewer parameters to train
- With large grammars, can reduce size of CFG grammar by over 90%
- Method described in (Dowding et al 2001)

Grammar compaction

- Three transforms, applied repeatedly until fixpoint is reached
  - "Absorbing": If non-terminal N occurs as LHS in just one rule, and RHS is all terminals, replace N everywhere with RHS
  - "Duplicate rules": Remove duplicated rules
  - "Duplicate rule groups": If the sets of rules for non-terminals N_1 and N_2 are the same, replace N_2 everywhere with N_1

Example

SIGMA → NP_SG

SIGMA → NP_PL

NP_SG → D_SG N_SG

NP_PL → D_PL N_PL

D_SG → the	D_SG → some
D_PL → the	D_PL → some

N_SG → sheep

N_PL → sheep

Example

SIGMA → NP_SG

SIGMA → NP_PL

NP_SG → D_SG N_SG

NP_PL → D_PL N_PL

D_SG → the D_SG → some

D_PL → the D_PL → some

N_SG → sheep (ABSORB)

N_PL → sheep (ABSORB)

Example

SIGMA → NP_SG

SIGMA → NP_PL

NP_SG → D_SG sheep

NP_PL → D_PL sheep

D_SG → the D_SG → some (DUPLICATE)

D_PL → the D_PL → some (DUPLICATE)

Example

SIGMA → NP_SG

SIGMA → NP_PL

NP_SG → D sheep (DUPLICATE)

NP_PL → D sheep (DUPLICATE)

D → the D → some

Example

SIGMA → NP (DUPLICATE)

SIGMA → NP (DUPLICATE)

NP → D sheep

D → the D → some

Example

SIGMA → NP

NP → D sheep

D → the D → some

Semantics

- Different possible approaches to semantics
- Approach 1(more general)
  - Compile plain CFG grammar
  - Reparse recognized words with unification grammar to get semantics
- Approach 2 (more efficient)
  - Compile annotated CFG grammar
  - Get semantics directly from recognizer

Using recognizer semantics

- Grammar Specification Language (GSL)
- Can build structured representations
  - Ordered lists
  - Attribute-value structures
- Can map restricted unification grammar semantics into GSL

Outline

- Overview
- Compiling unification grammars into speech recognizers
  - Unification grammar → CFG
  - Approximation using grammar specialization
  - Scalability
- Comparison of Regulus and other methods
- Using Regulus

Approximation using grammar specialization

- Large linguistically motivated grammars hard to compile
  - (Would be underconstrained anyway…)
- Use corpus-based grammar specialization to extract a reduced domain grammar
- Compile domain grammar into CFG

The general English grammar

- Loosely based on SRI Core Language Engine grammar
- ~175 unification grammar rules
- ~75 features
- Core lexicon, ~ 450 words

Overview of coverage (clauses)

- Clause types: declarative, Y-N questions, WH-questions, imperatives
- WH-movement of NPs, PPs, ADJPs and ADVPs
- Passives
- Impersonal subjects
- Embedded WH- and Y-N questions
- Relative and subordinate clauses
- Large number of sub-categorization types
- Adverbs

Overview of coverage (NPs and PPs)

- Conjunction of NPs, PPs, ADJPs and DETs
- Post-modification of NPs by PPs, ADJPs, relative clauses
- Pronouns
- Possessives
- Bare DETs as NPs
- Complex DETs
- Date, time and number expressions
- NPs as temporal adverbials

Grammars built so far

- Personal Satellite Assistant
- Home Automation
- Travel Deals
- Medical Speech Translator
- Intelligent Procedure Assistant
- Mobile Agents

Examples of coverage: Personal Satellite Assistant (PSA)

- Affirmative
- Go to flight deck
- Mid deck and lower deck
- Measure pressure
- What were oxygen and pressure one minute ago
- When did the temperature reach twenty degrees
- Go to the crew hatch and close it
- Close all three doors

Examples of coverage: Home Automation (HA)

- Is there a tv in the living room
- Which devices are turned on
- Turn on the kitchen light and the stove
- Dim the light to fifty percent
- Thank you

Examples of coverage: Travel Deals (TD)

- Holidays in paris under two hundred pounds
- I want something leaving from stansted
- In spain during may or june from gatwick
- Is there anything in italy before may tenth
- Give me a winter brochure
- Do you have three star or four star

Examples of coverage: Medical Speech Translator (MST)

- Do you often have headaches in the morning?
- Is the pain usually in the front of your head?
- Does the pain spread to your shoulder?
- Does red wine give you headaches?
- Are the headaches relieved by stress removal?
- How severe are the headaches?
- Is the frequency of your headaches increasing?

Examples of coverage: Intelligent Procedure Assistant (IPA)

- Next step
- Go back
- Go to step three point two
- No I said go to step five
- Set alarm for twelve minutes from now
- Record a voice note on step seven
- Delete voice note on step four point one
- Increase volume
- Say that again

Examples of coverage: Mobile Agents (MA)

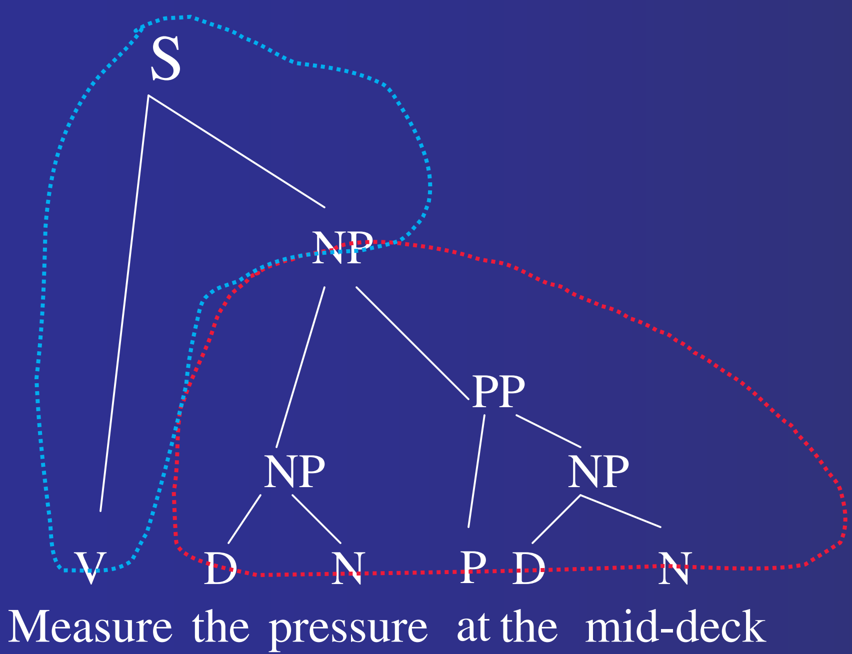
- Take a picture of me
- Boudreaux follow me now
- Return to the hab
- Start tracking my physiological sensors

Grammar specialization: Explanation Based Learning

- Macro-rule learning
- Corpus-based flattening of parsed examples to produce “larger” rules
- Learned grammar’s coverage is strict subset of original grammar’s coverage
- Coverage loss usually not serious
  - Specialized grammar often better in practice

Rule derivation using EBL

Training example



Derived Rules

S → V, NP

NP → D, N, P, D, N

Outline

- Overview
- Compiling unification grammars into speech recognizers
  - Unification grammar → CFG
  - Approximation using grammar specialization
  - Scalability
- Comparison of Regulus and other methods
- Using Regulus

Scalability

- How does it scale
  - … as general grammar gets bigger?
  - … as training set gets bigger?

Scalability with respect to size of general grammar

- General grammar built up by successively merging grammars for different applications
- Rationally reconstruct versions of general grammar for increasing numbers of applications
- Measure performance of PSA recognizers derived from increasingly large grammars

Data set used

- Personal Satellite Assistant data set
  - Collected in user tests of system
  - 10513 utterances (5394 training, 5169 test)
  - 38943 words
  - 27 speakers

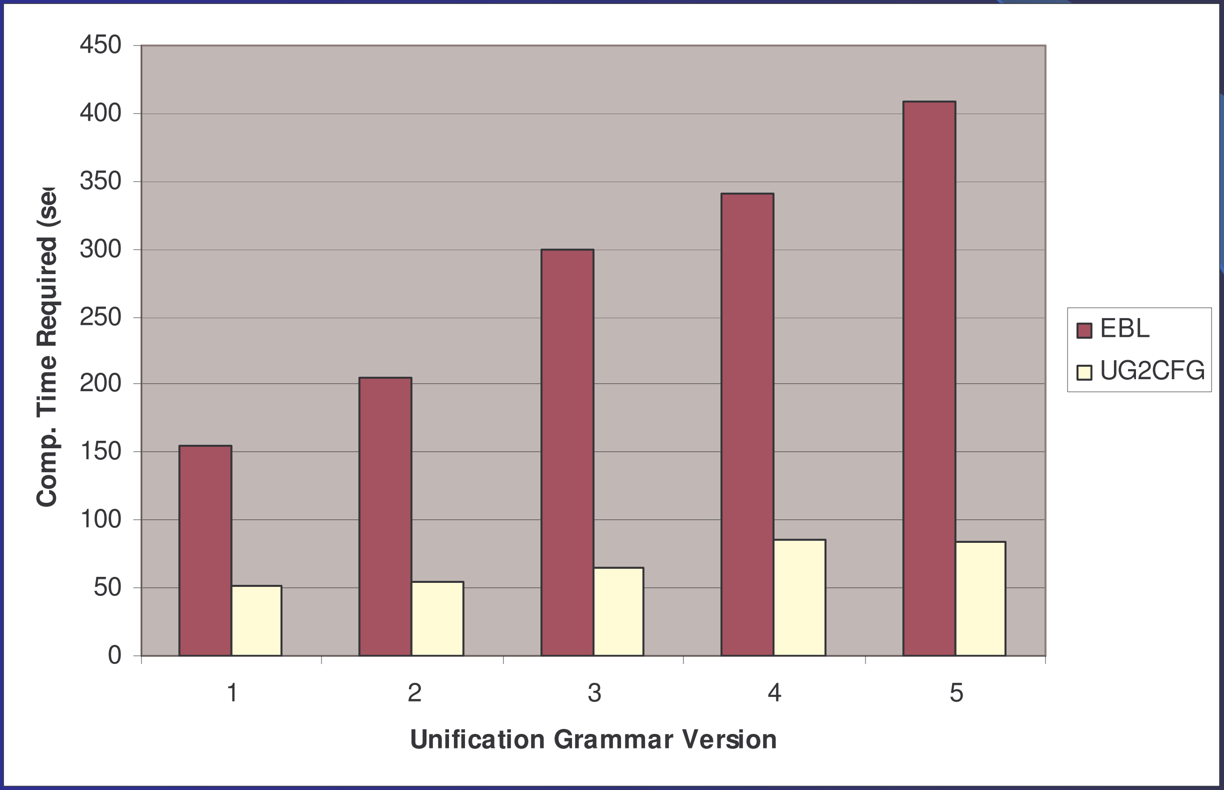
Parameters measured

- Compile-time
  - Time to perform grammar specialization
  - Time to perform UG → CFG compilation
  - Number of nodes in Nuance recognizer package
- Run-time
  - Word error rate (WER)
  - Proportion of utterances rejected (REJ)
  - Word error rate on non-rejected utterances (AWER)
  - Recognizer speed as multiple of real-time (xRT)

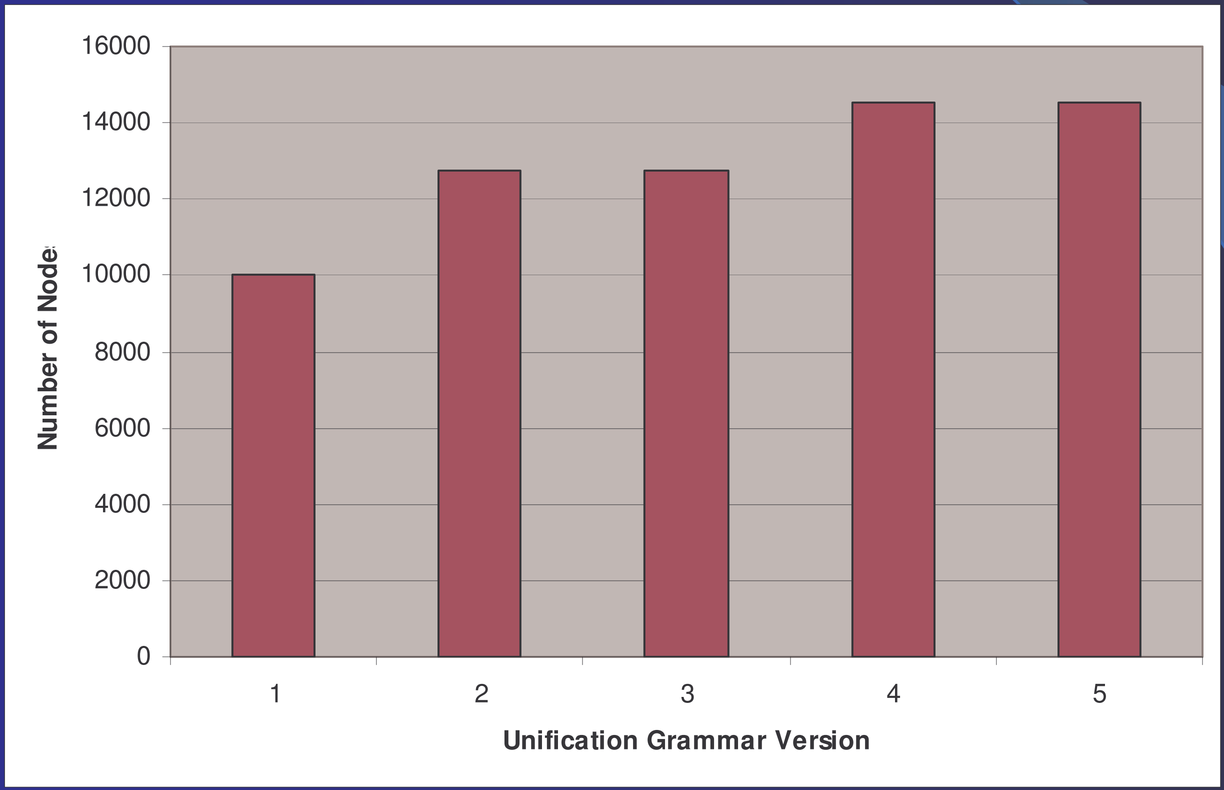
Sizes of different versions of general grammar

Version	Applications	#Rules	#Feats
1	PSA, HA	74	46
2	PSA, HA, TD	106	56
3	PSA, HA, TD, MST	127	64
4	PSA, HA, TD, MST, IPA	139	68
5	PSA, HA, TD, MST, IPA, MA	145	68

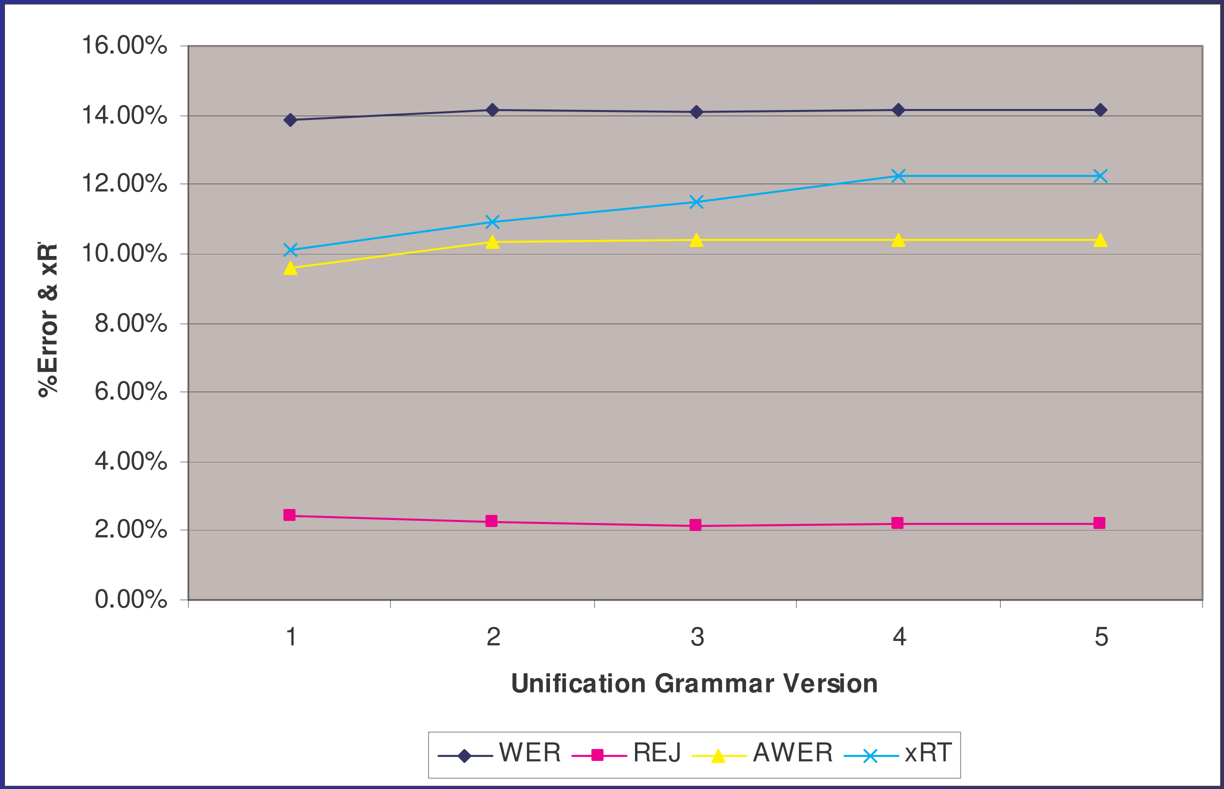
Scalability wrt size of general grammar: compile-time figures



Scalability wrt size of general grammar: size of recognizer



Scalability wrt size of general grammar: run-time figures



WER=word error rate

REJ=proportion rejected

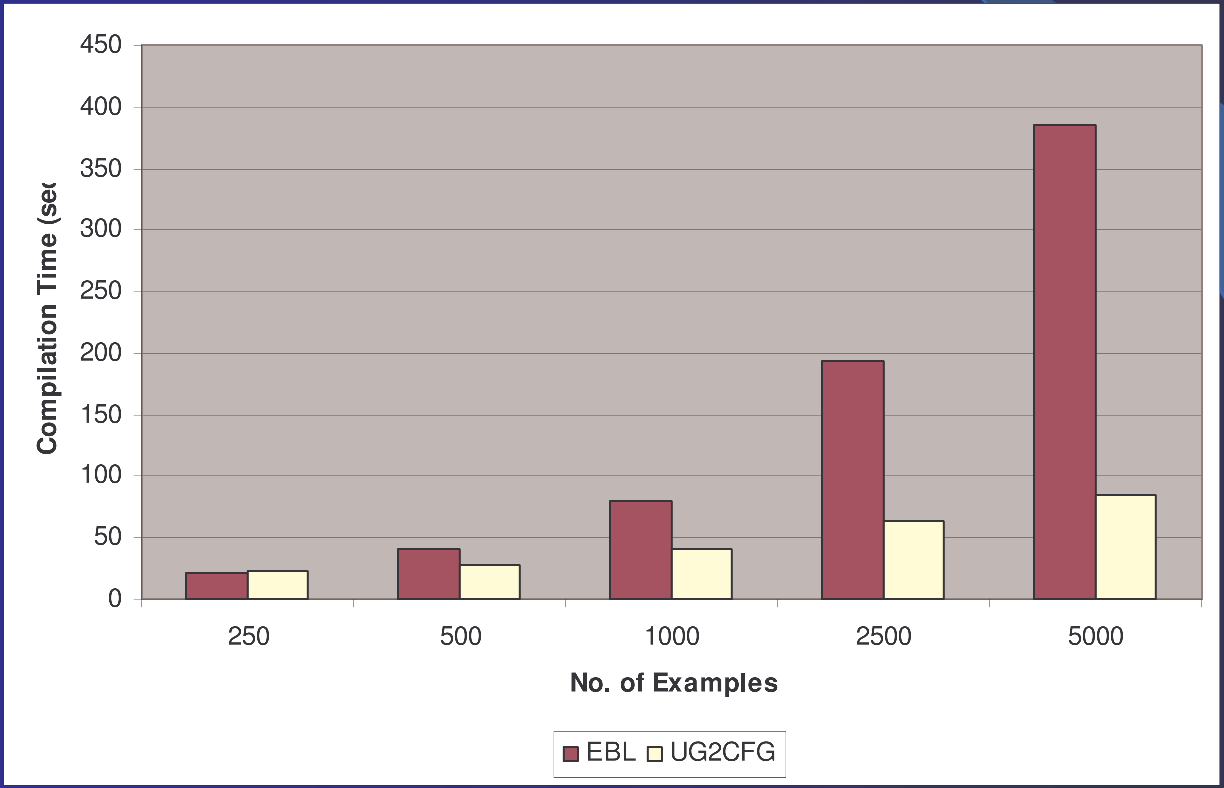
AWER=WER on accepted utterances

xRT=recognition speed (multiple of real time)

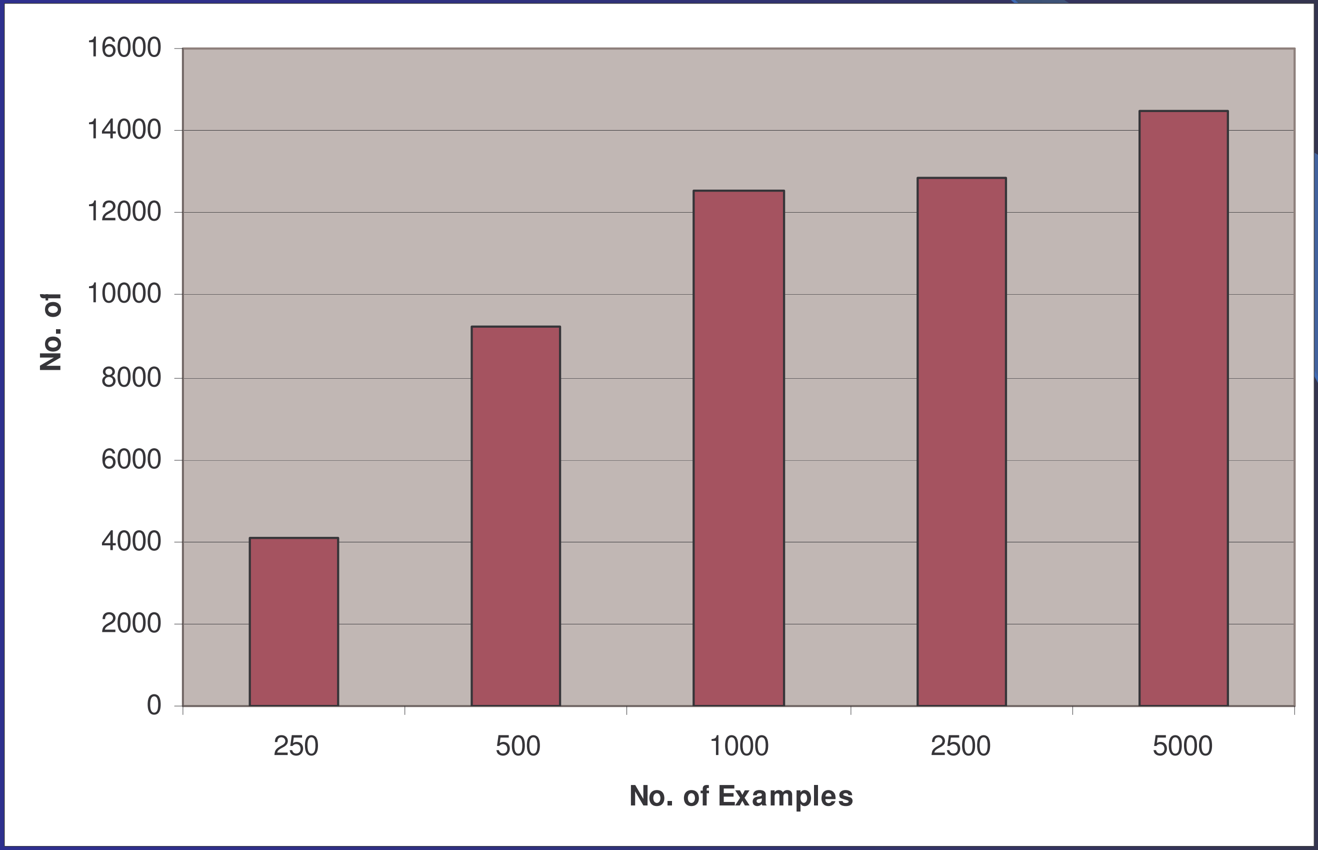
Scalability with respect to size of training set

- Train specialized grammars for PSA application
- Increase size of training set used to carry out grammar specialization

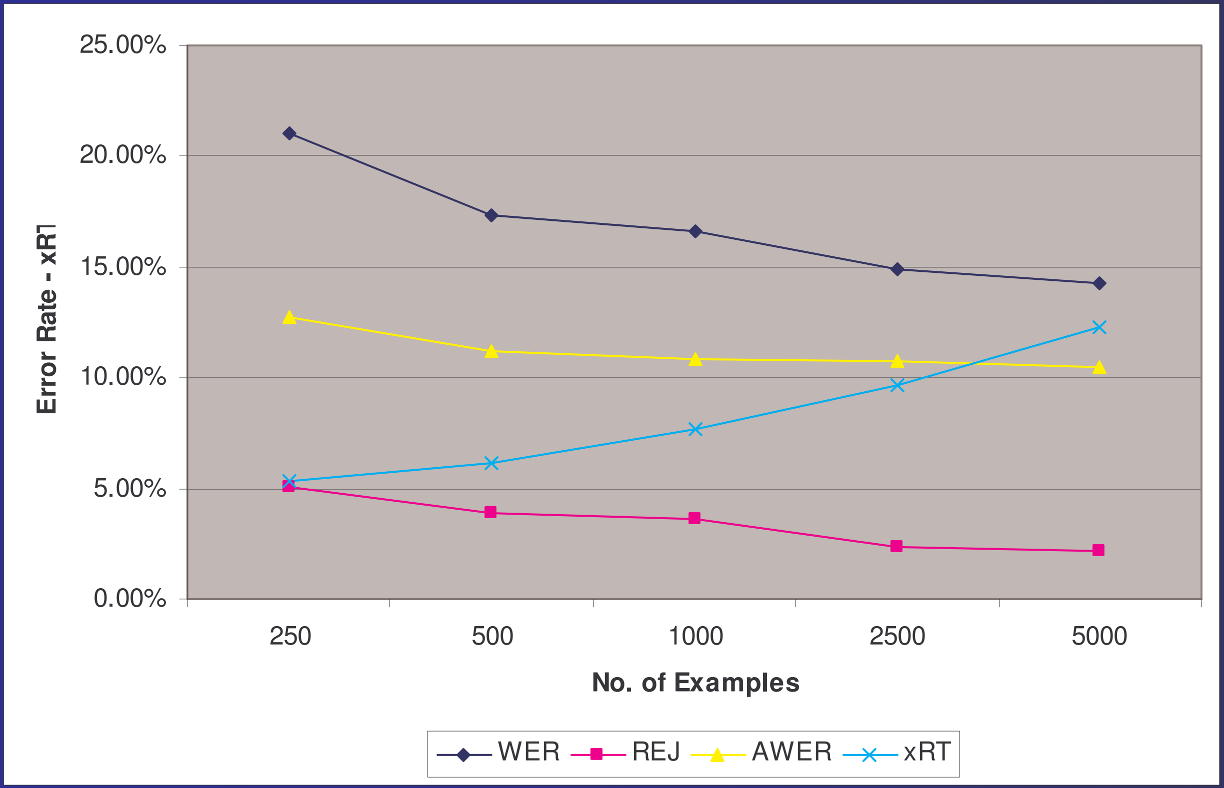
Scalability wrt size of training set: compile-time figures



Scalability wrt size of training set: recognizer size



Scalability wrt size of training set: run-time figures



WER=word error rate

REJ=proportion rejected

AWER=WER on accepted utterances

xRT=recognition speed (multiple of real time)

Summary of first half

- Overview
- Compiling unification grammars into speech recognizers
  - Unification grammar → CFG
    - Basic idea: exhaustive expansion
    - Refinements: interleaving, pre-processing…
  - Approximation using grammar specialization
  - Scalability

Coffee Break

Outline

- Overview
- Compiling unification grammars into speech recognizers
- Comparison of REGULUS and other methods
  – Comparison with hand-built grammars
  – Comparison with statistical/robust methods
- Using REGULUS

Comparison of specialized versus hand-coded language models

- Mobile Agents data
- Hand-coded grammar heavily optimized
  - Most challenging target for comparison
  - 60-70 rules, 2 weeks to build
- Specialized grammar done in one day
  - Mostly adding application-specific lexical items
  - Six grammar rules added

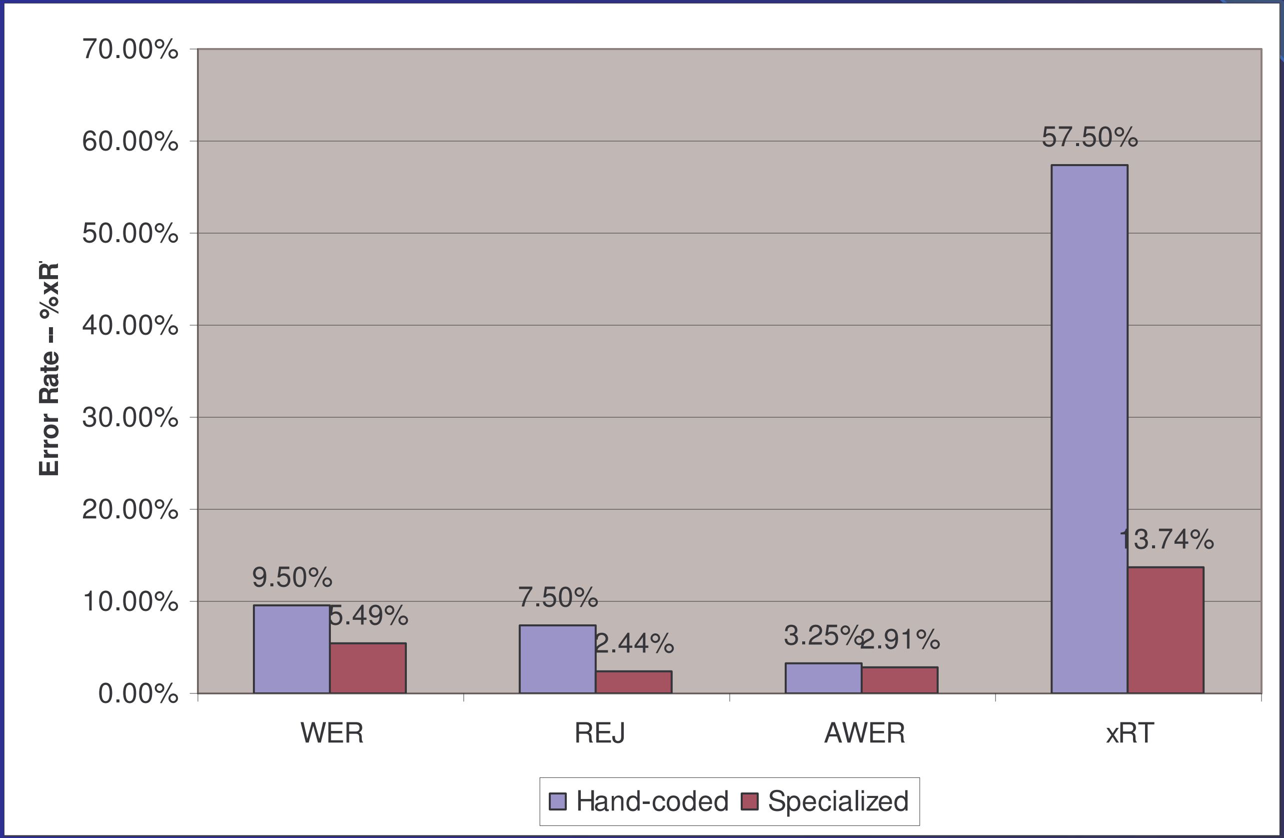
Training and Test material

- From September 2002 field test of Mobile Agents system
- 608 utterances (485 training, 123 test)
- 3535 words
- 8 speakers

Parameters measured

- Word error rate (WER)
- Proportion of utterances rejected (REJ)
- Word error rate on non-rejected utterances (AWER)
- Recognition speed as multiple of real-time (xRT)

Comparison of specialized versus hand-coded language models



WER=word error rate

REJ=proportion rejected

AWER=WER on accepted utterances

xRT=recognition speed (multiple of real time)

Why is the specialised version better?

- Specialization process tunes grammar efficiently
  - Faster recognition speed
  - Hand-tuning very time-consuming
- General grammar already covers many marginal constructions
  - Low-frequency constructions not always covered by hand-coded grammar

Outline

- Overview
- Compiling unification grammars into speech recognizers
- Comparison of REGULUS and other methods
  - Comparison with hand-built grammars
  - Comparison with statistical/robust methods
- Using REGULUS

Comparison with statistical/robust methods

- Build two versions of a system
- Compare performance
- Try to make comparison as fair as possible

System: Medical speech translator

- Open Source system built using Regulus
  - http://sourceforge.net/projects/medslt
- Limited-domain medical speech translation
- Doctor-patient examination domain
- One-way dialogue
  - Doctor can abort if recognition is bad
  - Patient responds non-verbally

Examples of coverage

- Do you often have headaches in the morning?
- Is the pain usually in the front of your head?
- Does the pain spread to your shoulder?
- Does red wine give you headaches?
- Are the headaches relieved by stress removal?
- Is the headache ever severe?
- Is the frequency of your headaches increasing?

Regulus (GLM) version

- Recognizer built using EBL grammar specialization
- Rule-based interlingual translation
- Regulus-based text generation
- TTS/concatenated wavfile speech output

Robust (SLM) version

- SLM-based recognizer
- Robust phrase-spotting parser
- Same translation module as in GLM version
- Same generation module as in GLM version
- Same speech output as in GLM version

Methodological issues

- Comparing a grammar-based recognizer with an SLM-based recognizer
  - Regulus lets us train the grammar-based version off the same data as the SLM
- Fair evaluation criteria
  - Evaluate on task performance, not artificial “semantic accuracy”

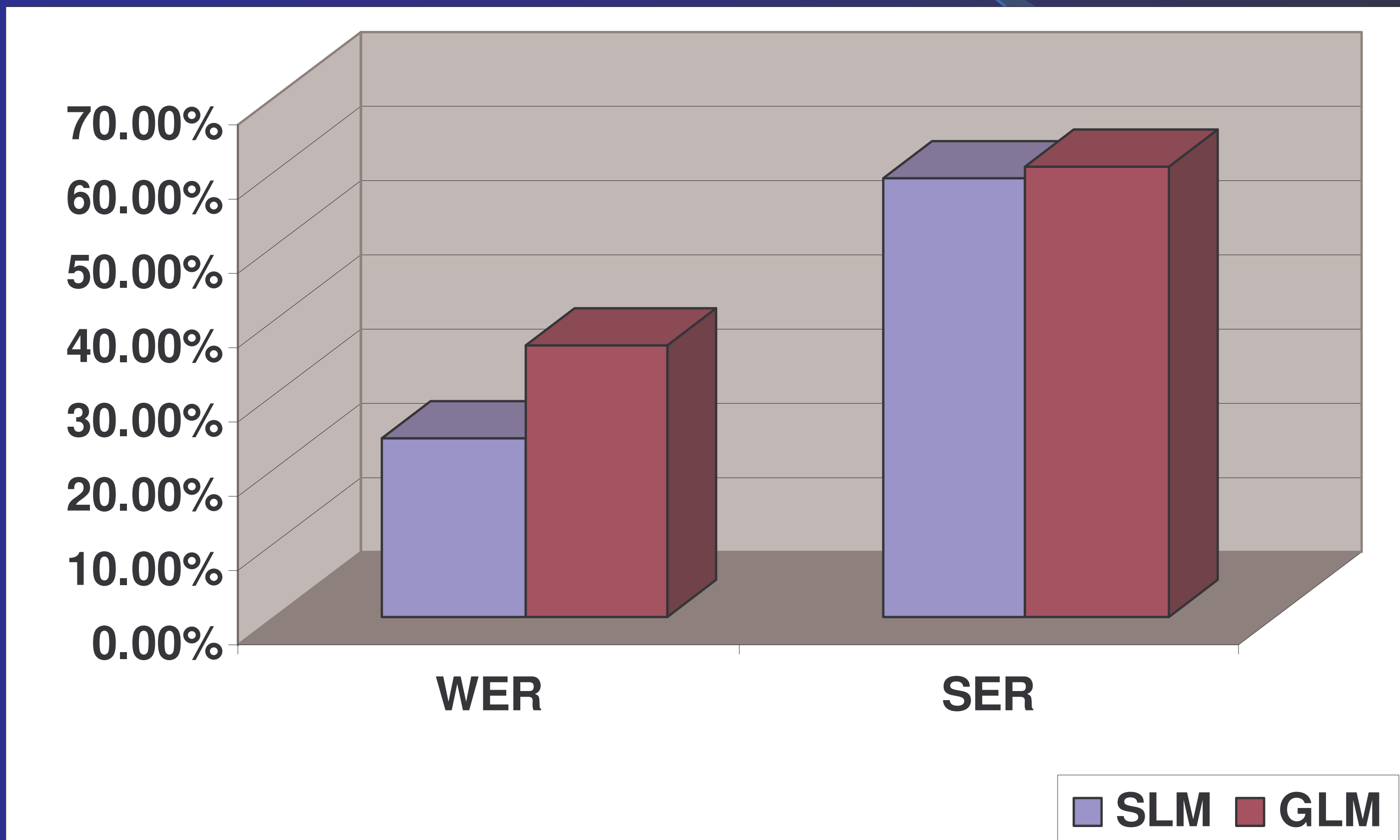
Training and test data

- Training data
  - 450 text utterances written by developers
- Test data
  - 524 spoken utterances collected from simulated use scenarios

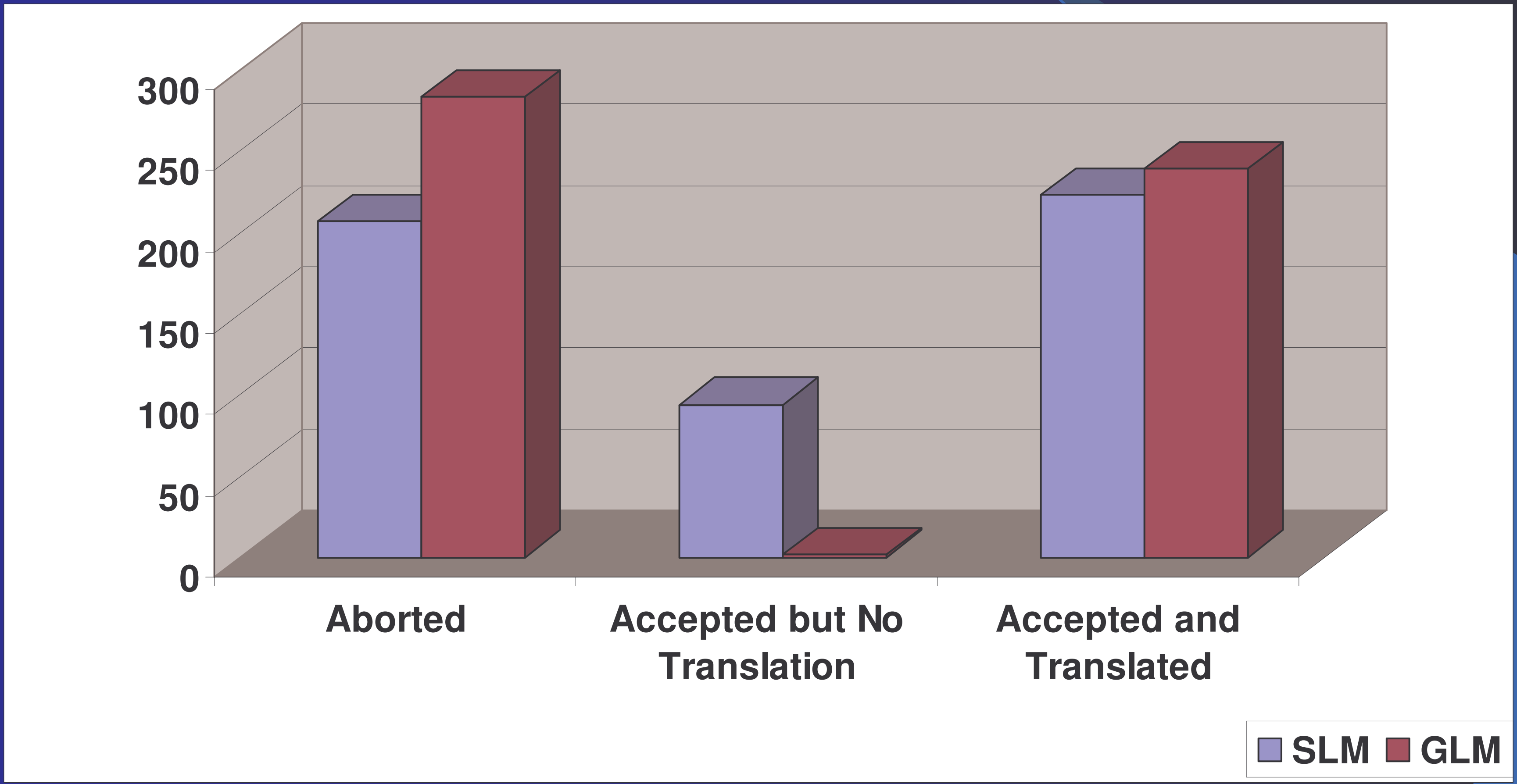
Experiments

- Process test data through both versions
- Judge recognition output for abort/accept
- Judge translations for accepted utterances
  - Three-point scale: good, ok, bad
  - Compare results across three judges

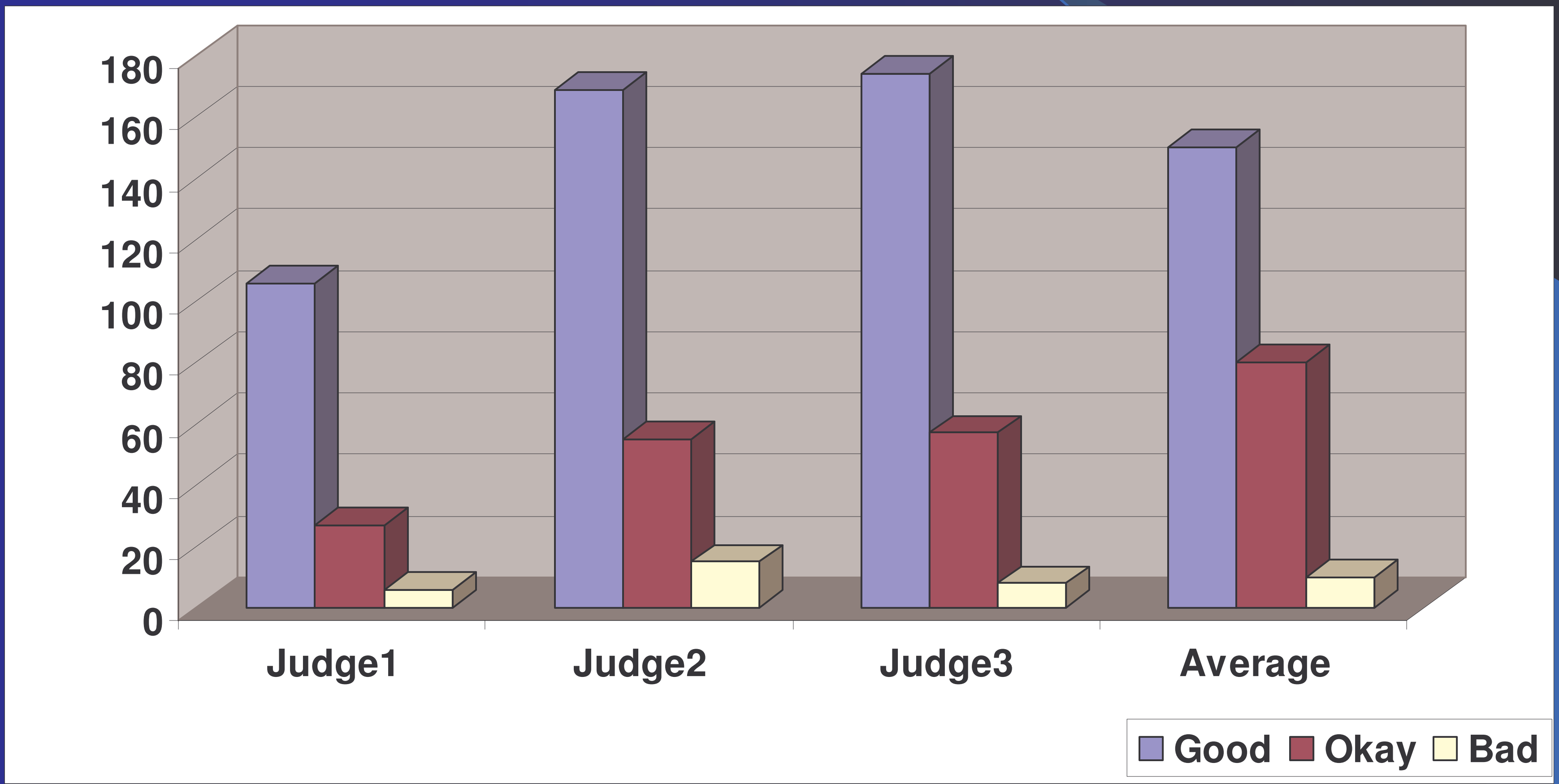
SER and WER in SLM and GLM versions



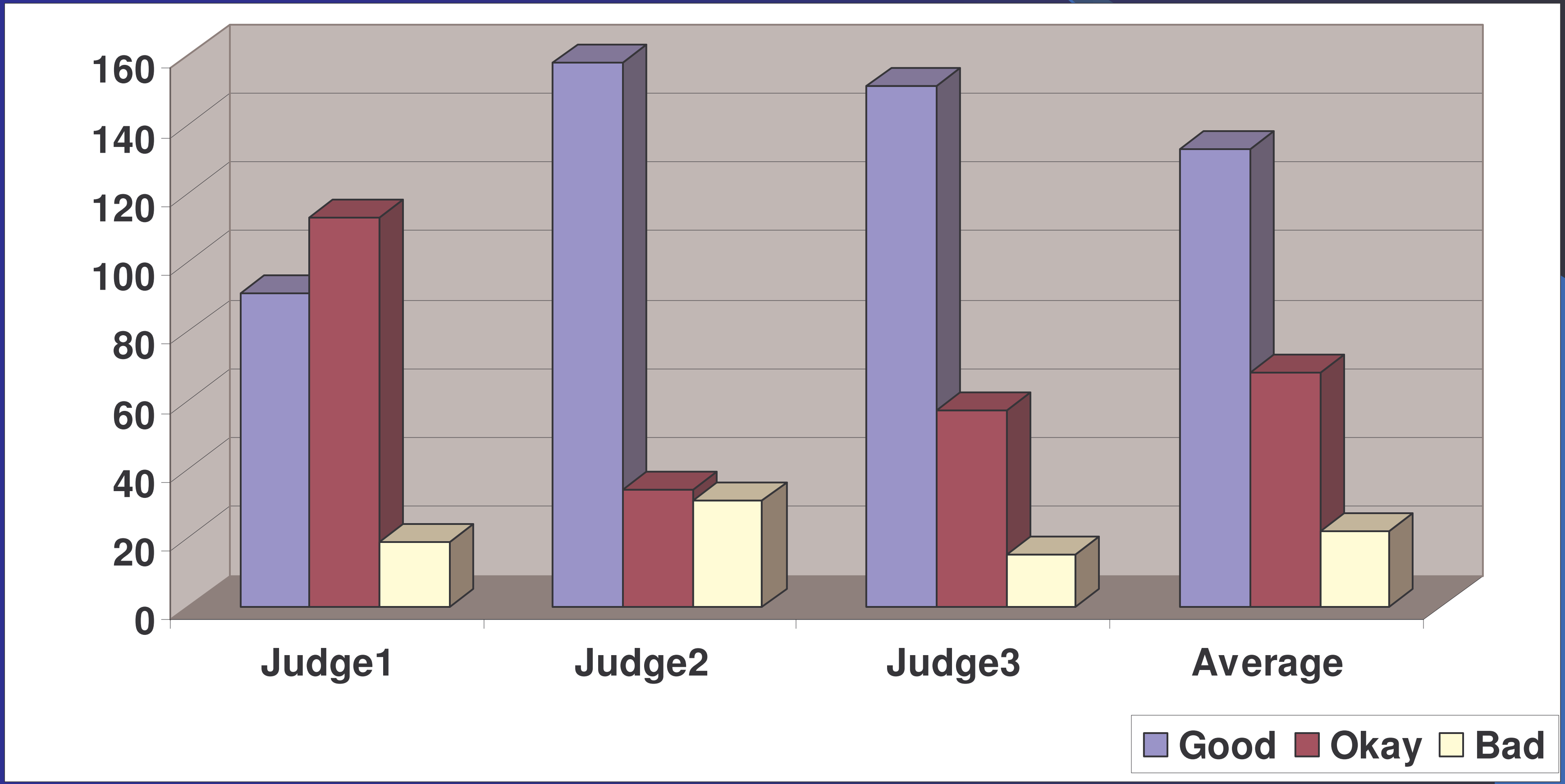
Breakdown of examples translated by SLM and GLM



Quality of translation with GLM version

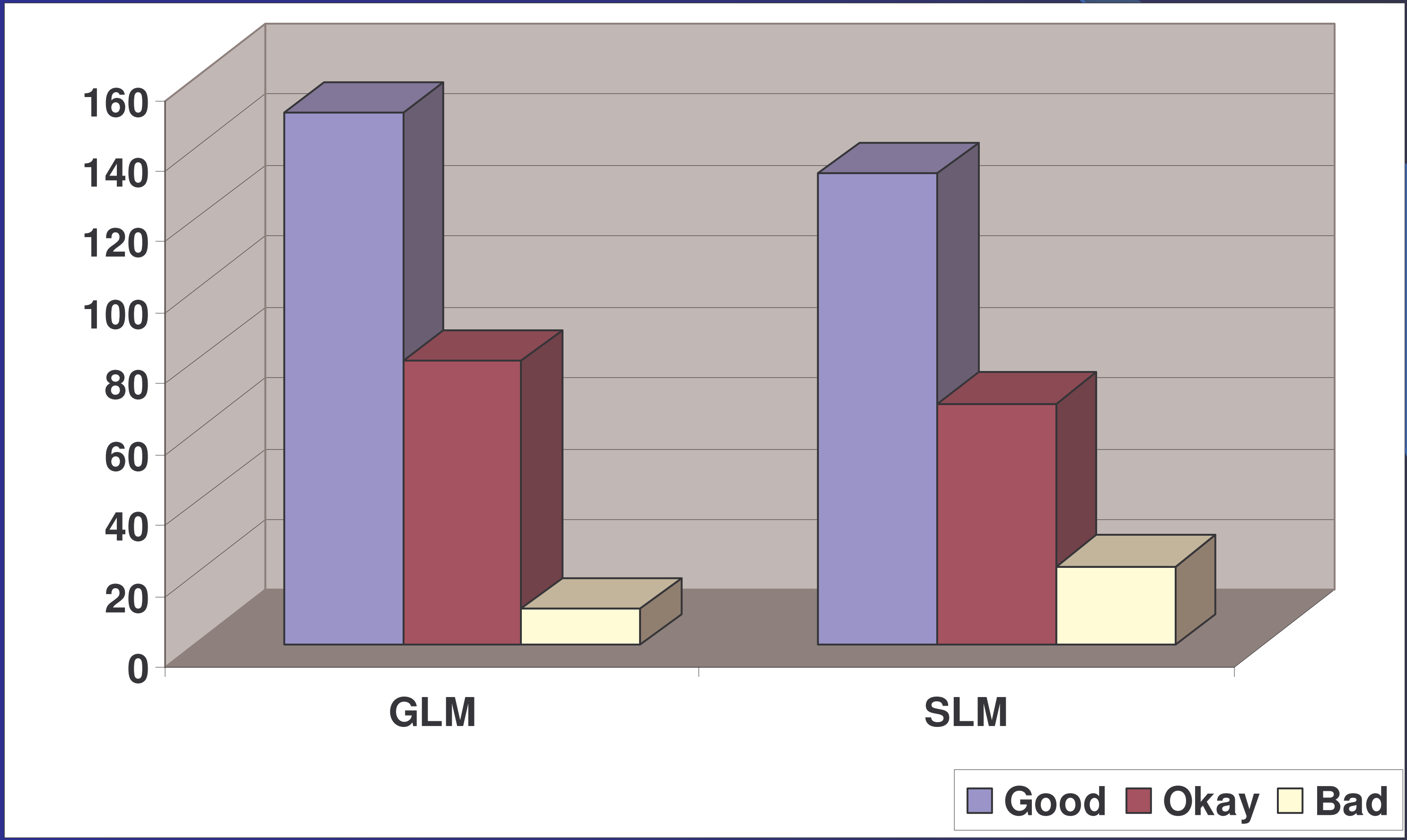


Quality of translation with SLM version



Quality of translation: Comparison of GLM and SLM

(translation judgements: averages)



Interpretation of results

- WER much better for SLM version
  - SER about the same
- Failed translations much more frequent
- Bad translations much more frequent
  - Many more translations all judges agree are bad

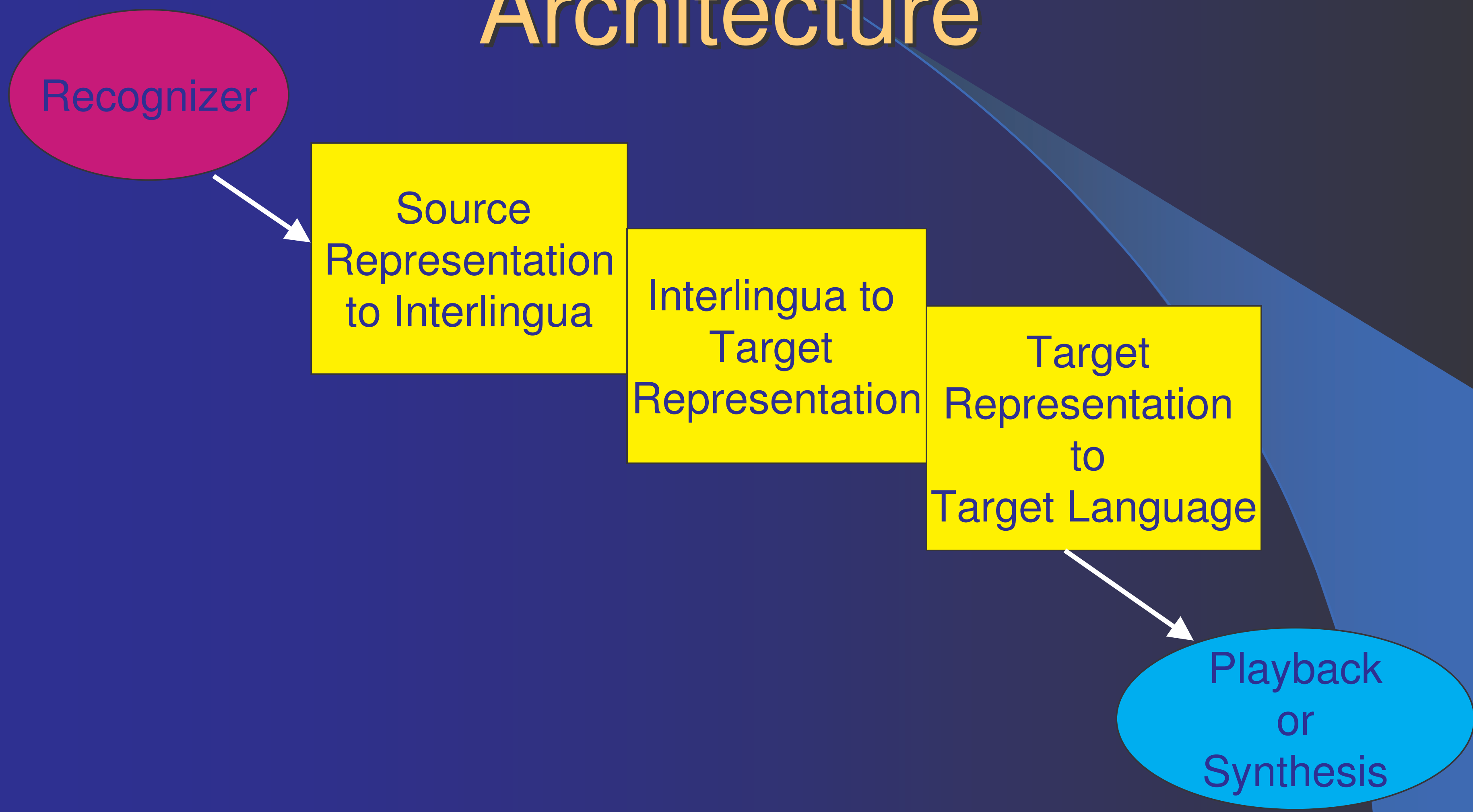
Why is the GLM better?

- Robustness doesn't help very much
  – "All or nothing" domain
- SLM version is much less predictable
  – Poor user experience

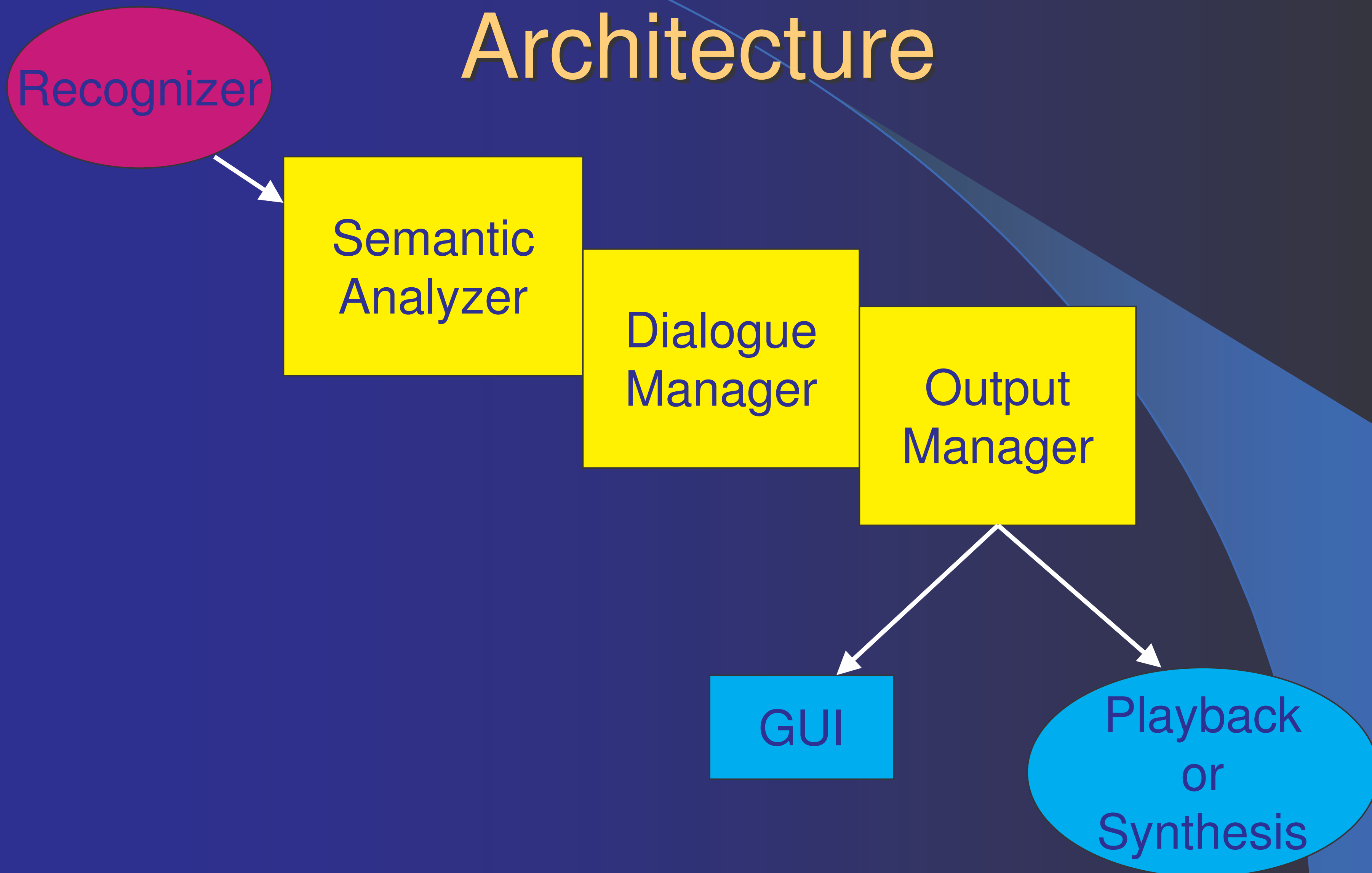
Outline

- Overview
- Compiling unification grammars into speech recognizers
- Comparison of REGULUS and other methods
- Using REGULUS

Speech Translation System Architecture



Spoken Dialogue System Architecture



Regulus components and functions

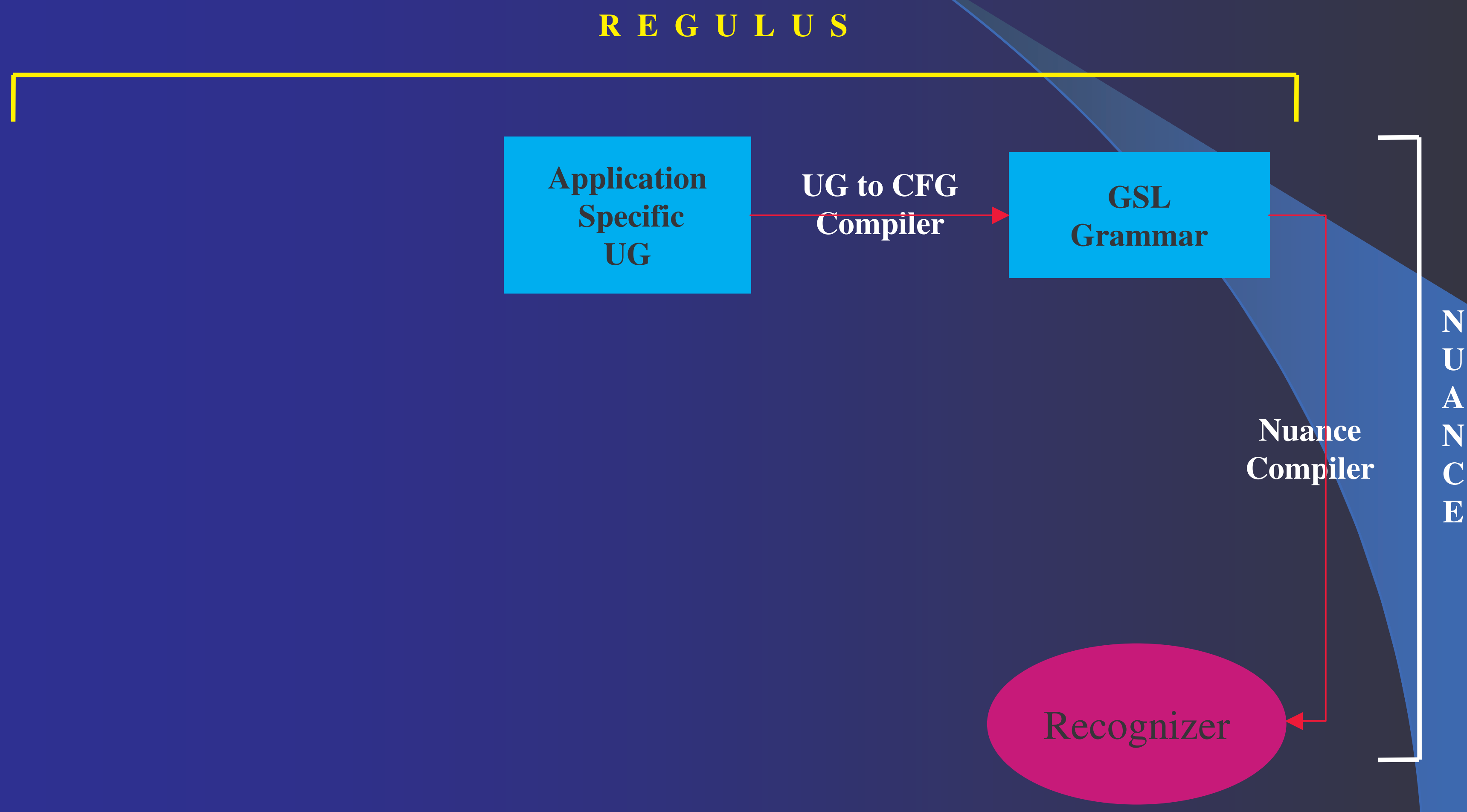
- Development environment
- Regulus → Nuance compiler
- Grammar specializer
- General grammars
- Parser generator
- Generator generator

Toy1

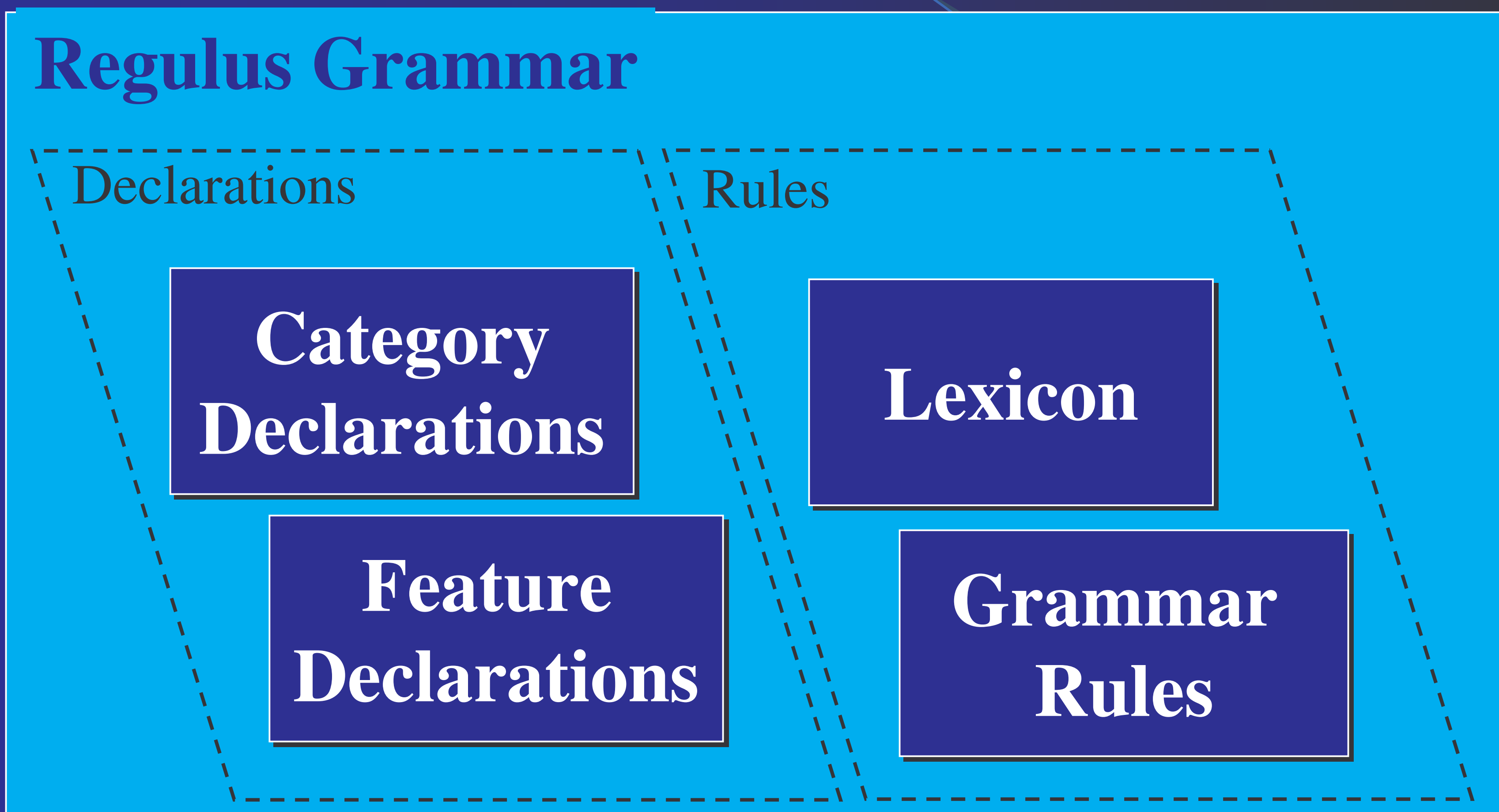
Toy1Specialized

ToySLT

Toy1: Building a recognizer



Grammar Components



Category Declarations

- Format

  category(CategorySymbol, [FeatureList]).

- Examples
  - Top level category

    top_level_category('.MAIN').

    category('.MAIN', [gsem]).
  - Lexical and phrasal categories

    category(yn_question, [sem]).

    category(noun, [sem, number, sem_np_type]).

Features

- Format
  - Feature value spaces

    feature_value_space(<ValueSpaceId>, <ValueSpace>).
  - Features

    feature(<FeatName>, <ValueSpaceID>).
- Examples

  feature_value_space(number_value, [[sing, plur]]).

  feature(number, number_value).

Lexicon

- Format

  <CategorySymbol>:<FeatValList> → lex item

- Examples

  noun:[sem=[[device, light]],
  sem_np_type=switchable\/dimmable,
  number=sing] --> light.

  verb:[sem=[[action, switch]], vform=imperative,
  vtype=switch, number=sing,
  obj_sem_np_type=switchable] --> switch.

Grammar rules

- Format

  Category → List of categories and/or lexical items

- Examples

  yn_question:[sem=concat([[type, query]],concat(Verb, concat(OnOff, Np)))] -->

  verb:[sem=Verb, vform=finite, vtype=be, number=N, obj_sem_np_type=n],

  np:[sem=Np, number=N, sem_np_type=switchable],

  onoff:[sem=OnOff].

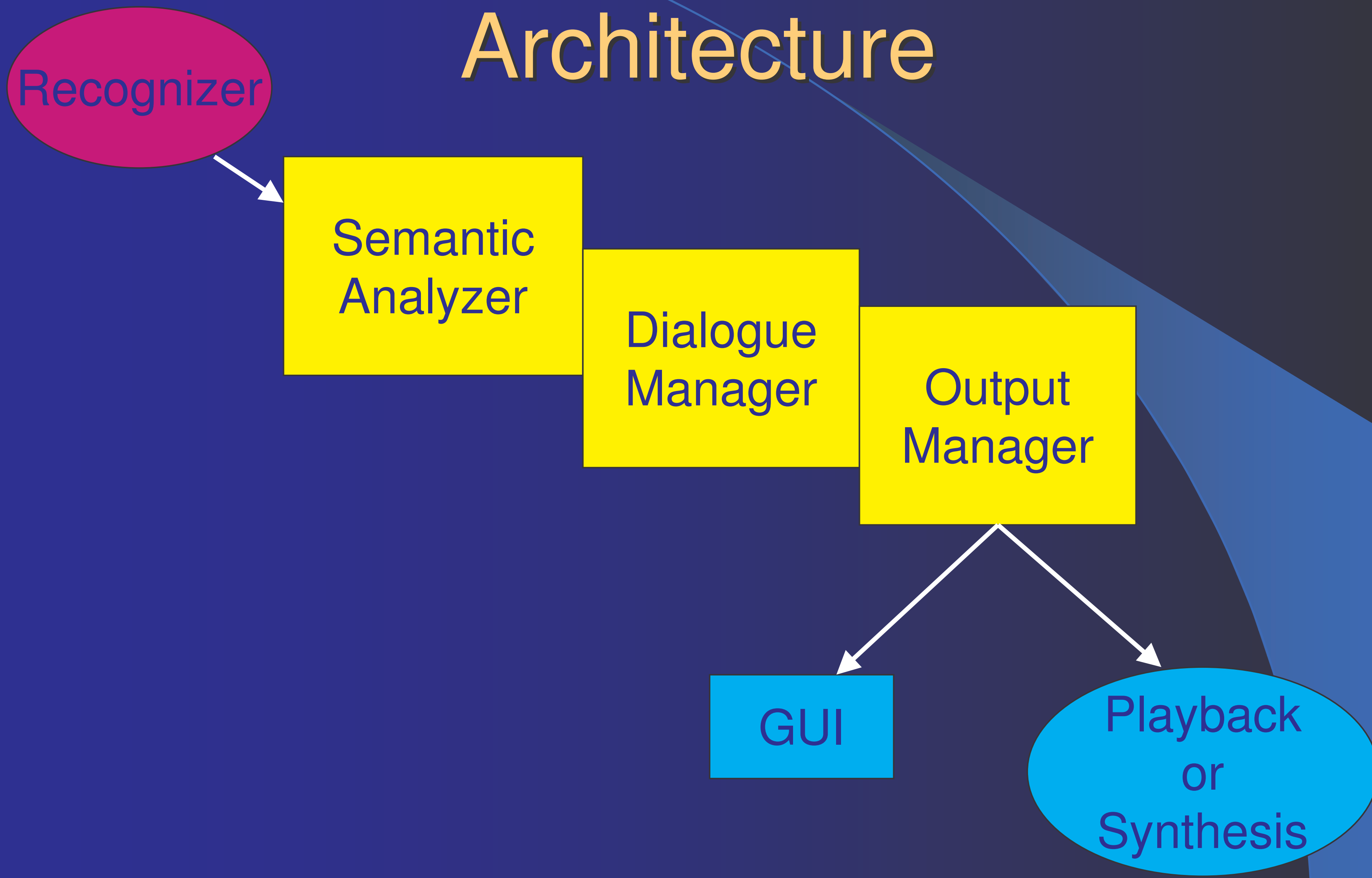
Development environment

- Key commands for Toy1
  - HELP
  - LOAD (Load current Regulus grammar in DCG and left-corner form)
  - NUANCE (Compile current Regulus grammar into Nuance GSL form)

Toy1: Building Recognizer

- UG → GSL in development environment
- UG → GSL using make
  - Alternative to doing it in the development environment
- Nuance compile

Spoken Dialogue System Architecture



Integrating with an application

- Toy1 application
  - Uses Regulus speech server
  - Minimal implementation of
    - Semantic Analysis
    - Dialogue Manager
    - Output Manager
  - Vocalizer TTS
  - Command line interface

Using the Regulus SpeechServer

- Recognition
  - Sends back Nuance results in same form as Regulus grammar
- Speech output
  - Sends request for TTS or for playing recorded wavfiles

Semantic Analysis

Language oriented semantics ⇨ Application oriented semantics

```
Recogniser representation: [[type,
 command],[action,switch],[onoff,on],
 [device,light],[location,kitchen]]
```

⇩

```
DM representation:
[command,device(light,kitchen,on,100)].
```

Dialogue Manager

Context + Dialogue Move → Context + DM Response

```
initial_state([
    device(light, kitchen, off, 0),
    device(light, living_room, off, 0),
    device(fan, kitchen, off, 0)]).
+
[command,device(light, kitchen,on,100)]
```

→

```
new_state([
    device(light, kitchen, on, 100),
    device(light, living_room, off, 0),
    device(fan, kitchen, off, 0)]).
+
device(light, kitchen, on, 100)
```

Output Manager

- DCG Template Generation

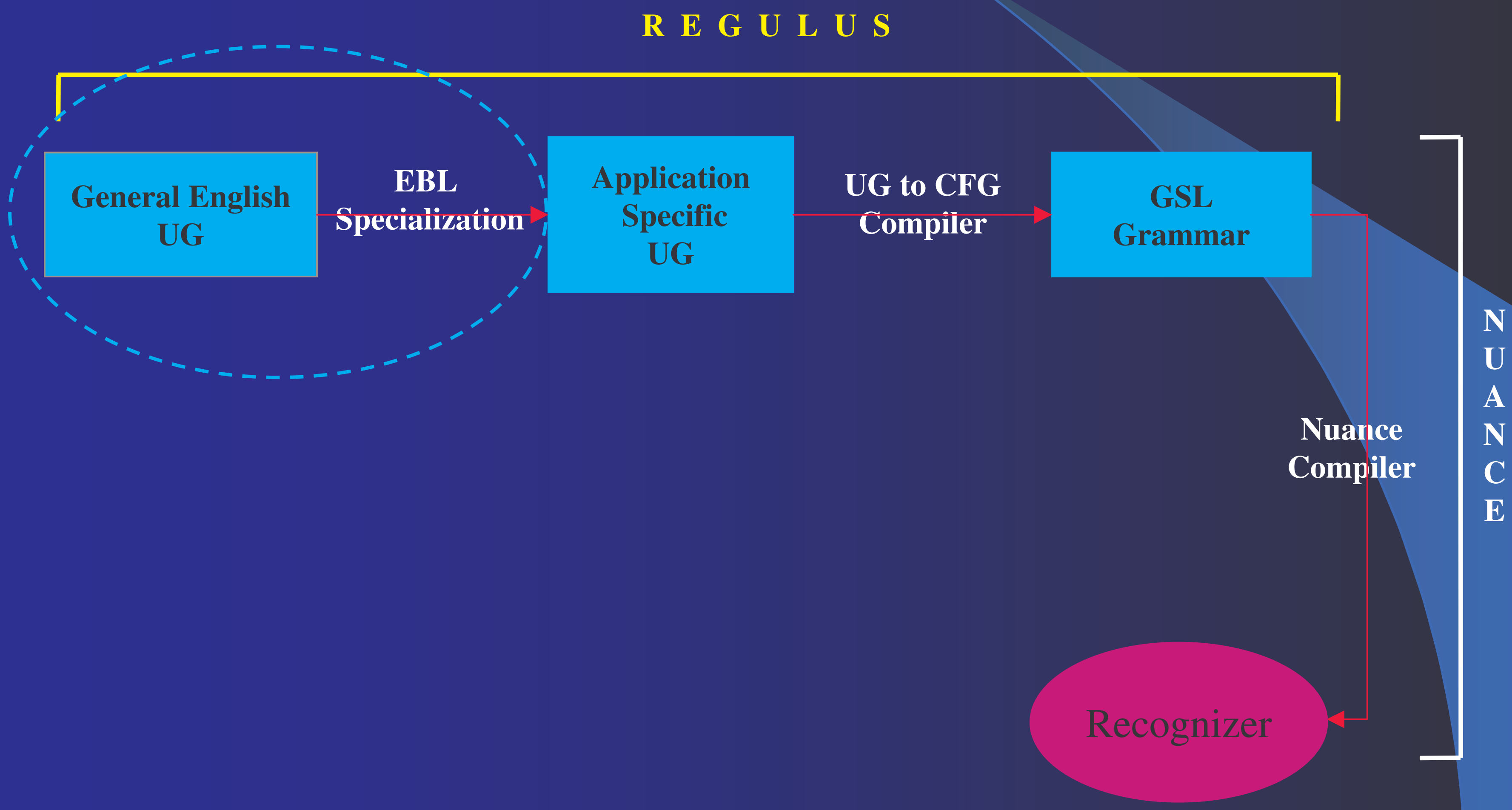
Abstract Response ⇨ Concrete Response

```
device(light,kitchen,on,100)
```

⇩

```
"the light in the kitchen is
on"
```

Toy1Specialized: EBL specialization



Specialization resources

- General English Grammar
- Training Corpus
- Domain Specific Lexicon

General English Grammar (1)

- Features: *vform, agr, nform, sem_n_type, obj_n_type …*

feature_value_space(agr_vals, [[1, 2, 3], [sing, plur]]).

feature_value_space(vforms, [[base, imperative, finite, ing, en, to, none]]).

…

feature(vform, vforms).

feature(agr, agr_vals).

…

General English Grammar(2)

- Lexicon: *on, the*

  p:[sem= @prep_sem(on_date), sem_pp_type=date, obj_sem_n_type=date] --> on.

  d:[sem=the_sing, agr=sing,wh=n,det_type=def,def=y,prenumber=n] --> the.

  d:[sem=the_plur, agr=plur,wh=n,det_type=def,def=y,prenumber=y] --> the.

General English Grammar

- Grammar Rules: *vp_v_p_np, np_d_n …*

```
vp:[sem= @vp_v_np_p_sem(Verb, NP, P),
   @vbar_feats_for_vp(Feats),
   takes_post_mods=y,
   gapsin=GIn, gapsout=GOut, elliptical_v=n] -->
vbar:[sem=Verb, subcat=nx0vplnx1,
     @vbar_feats_for_vp(Feats),
     obj_sem_n_type=ObjSem, obj_def=Def, obj_syn_type=ObjSynType,
     sem_p_type=PSem, elliptical_v=n],
p:[sem=P, sem_p_type=PSem],
np:[sem=NP, wh=n, nform=normal, sem_n_type=ObjSem,
    syn_type=ObjSynType, def=Def, takes_post_mods=n,
    @takes_no_pps, gapsin=GIn, gapsout=GOut, case=nonsubj,
    pronoun=n].
```

Training Corpus

- sent('switch on the light').
- sent('switch on the light in the kitchen').
- sent('switch the fan off').
- sent('dim the light in the living room').
- sent('is the light switched on').
- sent('is the light in the kitchen switched off').

Development environment

- Additional commands for Toy1Specialised
  - EBL_LOAD (Load current specialised Regulus grammar in DCG and left-corner form)
  - EBL_TREEBANK (Parse all sentences in current EBL training set into treebank form)
  - EBL_TRAIN (Do EBL training on current treebank)
  - EBL_POSTPROCESS (Postprocess results of EBL training into specialised Regulus grammar)
  - EBL_NUANCE (Compile current specialised Regulus grammar into Nuance GSL form)
  - EBL (Do all EBL processing: equivalent to LOAD, EBL_TREEBANK, EBL_TRAIN, EBL_POSTPROCESS, EBL_NUANCE)

Toy1Specialised: change corpus = change coverage

- With EBL, coverage can be changed by adding or deleting examples from the training corpus
- Doesn’t require linguistic expertise

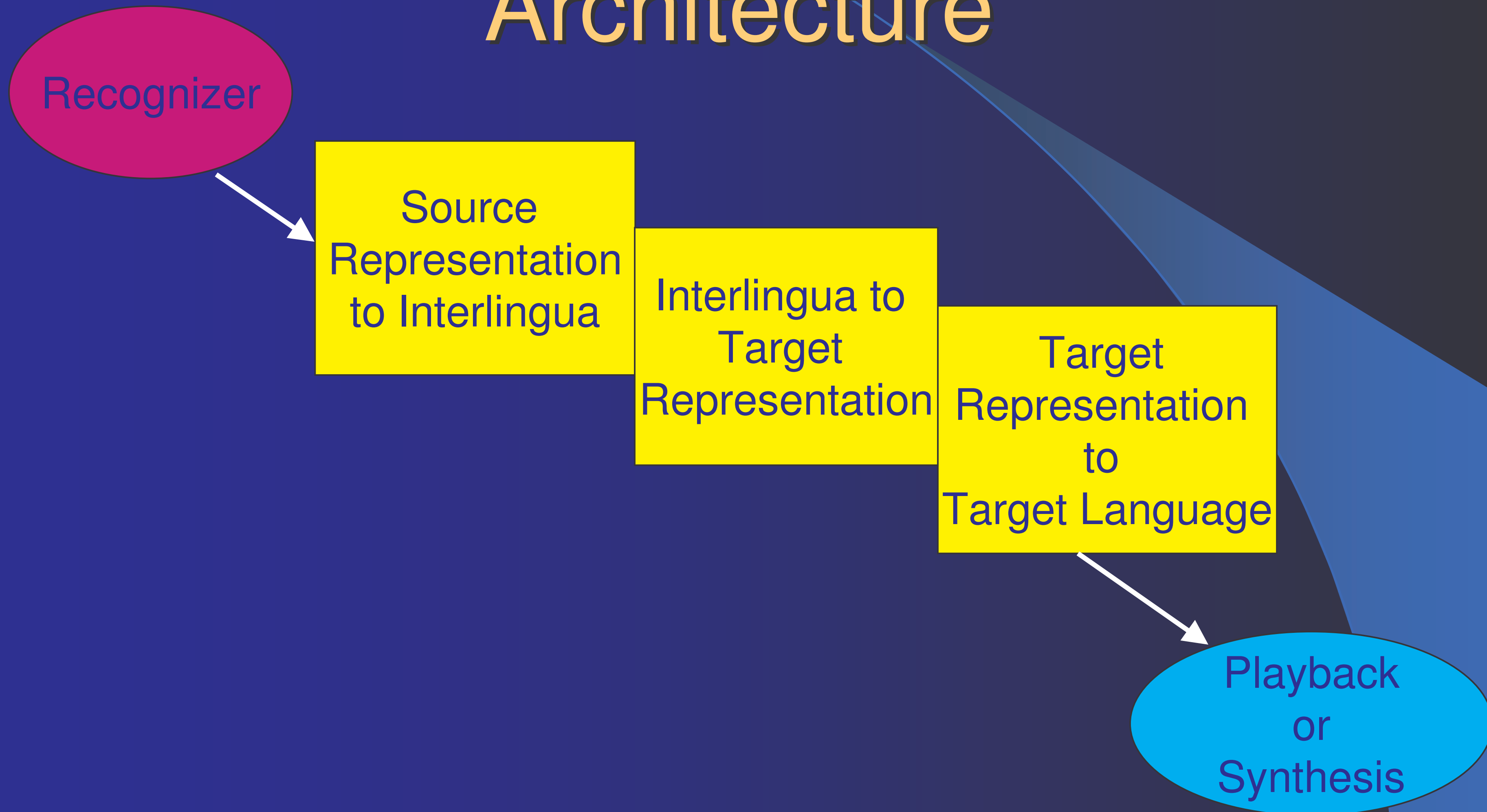
Changing coverage: Example

- Edit /Toy1Specialized/corpora/toy1_corpus.pl
- Development Environment-- “EBL” command does:
  - LOAD
  - EBL_TREEBANK
  - EBL_TRAIN
  - EBL_POSTPROCESS
  - EBL_NUANCE

ToySLT: Translation example

- Recognizer constructed with Regulus
- Connect to translation application
- Regulus based generation

Speech Translation System Architecture



Development environment

- Additional commands for ToySLT
  - LOAD_TRANSLATE (Load translation-related files)
  - TRANSLATE (Do translation-style processing on input sentences)
  - INTERLINGUA (Perform translation through interlingua)
  - NORMAL_PROCESSING (Do normal processing on input sentences)
  - LOAD_GENERATION (Compile and load current generator grammar)
  - GENERATION (Generate from parsed input sentences)

Integrating an application

- ToySLT application
  - Uses Regulus Speech Server
  - Minimal translation application
    - Source Representation to Interlingua
    - Interlingua to Target Representation
    - Target Representation to Target Language Using Regulus Generation
  - Vocalizer TTS

Source Representation to Interlingua

Recogniser Representation ⇨ Interlingua Representation

```
[[utterance_type,imp],[tense,
imperative],[pronoun,you],[action,
switch],[spec,the_sing],[device,
light],[prep,off]]
```

⇩

```
[[action,switch_off],[device,light],
 [type,command]].
```

Interlingua to Target Representation

Interlingua Representation ⇨ Target Representation

```
[[action,switch_off],[device,
light],[type,command]].
```

⇩

```
[[action,éteindre],[device,lampe],
[type,command]].
```

Target Representation to Target Language

- Regulus Generation:
  - Generator generator compiles regulus grammar into DCG optimized for generation

Target Representation ⟹ Target Words

```
[[action,éteindre],[device,lampe],
[type,command]].
```

⇓

```
Target words: "éteignez la lampe"
```

The Open Source Regulus project

- Where to find it
- Licensing terms
- Platforms/requirements
- Documentation and examples
- Installation

Where to find Regulus

- SourceForge www.sf.net
- Regulus Project Summary Page http://sourceforge.net/projects/regulus/
  - Stable releases available for download
  - Link for browsing the cvs repository
- CVS repository
  - Can check out current development version

Licensing terms

- Lesser GNU Public License (LGPL)
- Open Source license, BUT …
- … can incorporate Regulus into software products without these products becoming Open Source
  - Different from GLP license

Platforms/requirements

- Windows 2000/XP, SunOS/Solaris
  - Cygwin recommended if using Windows
- SICStus Prolog version 3.10 or newer
- Nuance 7.0 or newer
- 256 MB or more
- 1 GHz or more recommended

Documentation and examples

- Documentation (in HTML): /Regulus/doc/RegulusDoc.htm
- Example grammars/systems: /Regulus/Examples
    - Toy1
    - Toy1Specialised
    - ToySLT
    - PSA

Installation

- Unpack zipfile
- Set environment variables
- Install other software if necessary
  – SICStus Prolog
  – Nuance
  – Cygwin

Summary and conclusions

- Can derive recognizers for multiple applications from one general grammar
  - Faster development times
  - More reusable
- Good scalability properties
- Competitive with
  - Hand-coded grammars
  - Robust/statistical methods
- Available on Open Source platform
- Regulus Book 2005